Distributed Job Scheduling in a Peer-to-Peer Video Recording System

Curt Cramer, Kendy Kutzner, and Thomas Fuhrmann
Institut für Telematik, Universität Karlsruhe (TH)
{cramer|kutzner|fuhrmann}@tm.uka.de

Abstract: Since the advent of Gnutella, Peer-to-Peer (P2P) protocols have matured towards a fundamental design element for large-scale, self-organising distributed systems. Many research efforts have been invested to improve various aspects of P2P systems, like their performance, scalability, and so on. However, little experience has been gathered from the actual deployment of such P2P systems apart from the typical file sharing applications. To bridge this gap and to gain more experience in making the transition from theory to practice, we started building advanced P2P applications whose explicit goal is "to be deployed in the wild".

In this paper, we describe a fully decentralised P2P video recording system. Every node in the system is a networked computer (desktop PC or set-top box) capable of receiving and recording DVB-S, i.e. digital satellite TV. Like a normal video recorder, users can program their machines to record certain programmes. With our system, they will be able to schedule multiple recordings in parallel. It is the task of the system to assign the recordings to different machines in the network. Moreover, users can "record broadcasts in the past", i.e. the system serves as a short-term archival storage for TV programmes, too.

Since we want the system to be evaluated by a large number of users, we decided to build it as a plug-in for the popular and widely deployed Linux Video Disk Recorder (VDR) software (http://www.cadsoft.de/vdr/). VDR already offers the basic recording functionality, thus we can focus on the aspect of cooperatively scheduling recordings. No definite results can be discussed yet, as the work presented here is in progress. Nevertheless, we are already able to point out some important design issues and open research questions.

1 User Interface Requirements

Many P2P protocol improvements and new applications have been proposed. However, it seems that, apart from the well-known filesharing applications and a few others, notably Skype (http://www.skype.com), little expertise in bringing P2P applications to the "real world" has been developed. We believe that existing solutions might need to be adapted and new problems might arise when actually deploying P2P applications. In this paper, we propose a P2P overlay application to be adopted by a broad user base for studying these particular aspects. We begin with user interface aspects in this section and describe their implications for the system in the next section.

We are currently designing a P2P networked satellite video recorder. This device can

be operated like a standard personal video recorder (PVR) that receives an MPEG video stream via digital video broadcast (DVB) and stores it on a hard disk. By sharing their resources with the networked peers, devices additionally enable their users to issue multiple requests in parallel, even if the user's device is already occupied with recording a TV station. Requests and the resulting recordings are shared among the devices. Moreover, a user may decide to "record" a movie even after it has been broadcast, provided that other devices in the network still have a copy of that movie left. Concerning the recording devices, one can think of three basic types:

1. Stand-alone devices similar to the commercial PVRs sold today. These set-top boxes are operated by an on-screen display (OSD) using the attached TV set.
2. Hybrid devices that use a desktop PC's window manager both for operating the device and for displaying the video content.
3. Remote access devices that are accessible by a web browser. Video content can then be downloaded or streamed to the remote device after recording it.

Though convenient, the latter version is questionable for its susceptibility to copyright infringements. We therefore focus on stand-alone and hybrid devices which, by definition, only share content with equal peers that would have been able to legally acquire the content.

Human users issue requests based on the Electronic Program Guide (EPG) which is broadcast in parallel to the video data. As it is our explicit goal to build a deliverable which is to be widely accepted, we strongly believe that the additional operating complexity imposed on the users by the networking aspect must be kept as low as possible (according to the "Keep it simple, stupid!" (KISS) design principle). To this end, we came up with the following simple operating metaphor:

1. Users can classify recording requests as a *must*. These requests occupy the respective user's device exclusively and are locally executed, provided that the device is in working order and sufficient disk space is available. This exactly is a classic PVR's mode of operation.
2. Other requests are done at the *should* level, i.e. they may be automatically delegated to other devices in the network. If these devices fail or currently serve other requests, those recordings might fail. The system is to maximise the number of satisfied *should* requests in a best effort manner.

Since no other requests can be served by a device occupied with a *must* request, it should disencourage its owners from issuing such requests, e.g. by requiring an extra confirmation. We expect users to actually refrain from submitting *must* requests once they get used to the system successfully serving a large fraction of their *should* requests.

Overlapping *should* requests have to be ordered by priorities. This is illustrated to the user by the requests being piled up along a timeline. Requests in the upper row are *must* requests. Rows below this "must border" contain *should* requests ordered descending by their priorities. Users need not understand the scheduling, but simply drag requests between the priority levels to resolve collisions according to their preferences (e.g., "Drag important things to the top."). It is the system's task to ensure that higher-prioritised jobs are scheduled before lower-prioritised jobs. Depending on the employed scheduling algo-

rithm, the system could even provide graphical feedback to indicate the request satisfaction probability (e.g., "Drag important things into the green area and keep your fingers crossed for the rest.") As will be discussed later, this user interface designed with the KISS principle in mind also has implications for the overall system behaviour.

2 The Distributed Job Scheduling Component

DVB streams are continuous MPEG streams typically not identifying individual broadcasts, i.e. recordings are only scheduled by their start and end times. To reduce the complexity of the scheduling problem, we divide time into discrete units, e.g. blocks of five minutes. This is invisible to the users. Note that glueing blocks from different recorders requires the inspection of overlapping blocks, since clocks might not be synchronised at sub-second precision. Thereby, we can always glue exactly at the frame group boundaries.

The priority assignment generated from the user's requests results in a priority function which maps a channel identifier and a block-time to a priority level. It effectively is the priority function which is submitted to the system by each user. The result of the scheduling algorithm is an assignment of channel IDs and block-times to the participating recorders.

The problem of assigning recordings to recording devices can be seen as a combinatorial optimisation problem with two objective functions. First, we want to *maximise* the number of requests fulfilled by the system. Second, the number of channel switches between subsequent recordings on a single recorder is to be *minimised* to avoid problems with glueing recordings from different recorders. Since users may revoke requests or enter new requests at any time, the scheduling algorithm has to be dynamic and preemptive, i.e. the schedules have to be adapted whenever requests change.

To actually solve the combinatorial optimisation problem in a distributed and P2P fashion, we employ an aggregating Distributed Hash Table (DHT) with proximity preference. The overlay address space consists of the product space of block-times t and station identifiers i. With an unstructured overlay implementing this space, requests had to be flooded. This is unscalable since there will be a large volume of requests, and for scheduling, it suffices to have local knowledge. Therefore, a structured P2P overlay or a DHT is the superior approach to unstructured overlays.

To reduce load problems arising from many nodes issuing the same request, aggregation is used: The nodes along the path between a requesting node k and $N_{(t,i)}$, which is the node responsible for managing point (t, i) in the space, aggregate similar requests.

In this project, we envisage two different approaches for comparison:

Approach 1: Each node determines partial schedules *for the (aggregated) requests that are routed to it*. This is a classical approach, where the problem is partitioned among nodes, but each partition is solved by a designated node alone. To achieve the two stated objectives, the node has to communicate with the nodes surrounding it in the overlay address space in order to correct and adjust its decisions based on the neighbours' decisions. This approach is reminiscent of the *Distributed Constraint Satisfaction Problem* [YDIK98].

Approach 2: Each recorder decides *for itself* which station to record based upon the aggregated requests and prior scheduling results in the DHT. Here, the heart of the scheduling algorithm lies in the way results are aggregated in the DHT: high priority requests contribute more and the farther they propagate from their origin, the more attenuated contributions get. Each recorder can thereby determine those channels that are urgently wanted (in its neighborhood) but not (sufficiently) served. By announcing its decision to record a channel, it suppresses others from doing the same. If recorders continuously and independently make and announce their decisions, the system will reach an equilibrium state, which yields the overall schedule. This approach is reminiscent of algorithms inspired by biological systems.

Besides this scheduling functionality, the DHT also supports the retrieval of recorded content: Once a block has been recorded, pointers to (a small number of) stations that could provide this block to others are cached along the path in a soft-state manner. Content must be re-announced regularly as long as a device provides a copy.

3 Open Issues

The goal of this project is not only to compare, e.g., the two described scheduling approaches for studying proximity-awareness and aggregation, but to also gather experience with the actual deployment of such P2P systems. In particular, we want to answer questions like:

- How do the two scheduling approaches compare under real request patterns?
- Is proximity sufficiently accounted for by the aggregation process?
- What are good aggregation functions for the the second approach?

We believe that these questions can only be answered by a project that aims at real-world settings. There, interesting facets of the problem surface from the social behaviour of the users. (E.g., the model of the prisoner's dilemma with many participants.) We designed the user interface to diminish selfish user behaviour, but it is unclear whether this suffices:

- Would it be more effective to let the system assign priorities to the requested recordings depending on the request frequencies?
- Will users start to modify their devices to gain unfair shares of the system?
- How could counter measures be included into the system from the start?

Furthermore, as user behaviour is unpredictable and the individual recording devices may show significant downtimes (e.g. for cost reasons), the scheduling component has to provide some reliability. This could, e.g., be done by replicating requests on several machines. Open questions with this are:

- Should the replication factor scale with the request frequency or not?
- If the replication factor is to be defined as a function of the request frequency, we need a formal definition of this optimisation problem. What is the optimal solution to this, i.e. how does this function look like?
- How can replicas be located? How can the replication factors be calculated in a

distributed system? Some kind of aggregation will be needed for this.

A first version of the system is currently in the making. We hope to be able to deploy it in late summer. Then, we will have the means to study the questions posed above.

4 Related Work

Like our project, the *Share it!* project [WMM03] investigates the goal to create a peer-to-peer networked digital video recorder. However, their system was not designed to offer the unique feature of distributed recordings. Their goal is to define a platform for exchanging content after it has been independently and individually recorded by the different PVRs.

The *UP-TV* project [SSB04] considers distributed recordings in a set of digital video recorders. Users and recording devices connect to their local servers, which are organised in clusters and are running a centralised scheduling algorithm. Apart from this centralisation, the project differs from ours in two points: First, our system aims at a large-scale networked system. UP-TVs clusters are interconnected, too, but they do not cooperatively schedule their recordings. Each cluster only schedules local recordings. Second, the utility values or preferences assigned to recordings by the users are not restricted in UP-TV. This again is an instance of the above mentioned multi-person prisoner's dilemma.

5 Conclusion

In this paper, we outlined a fully distributed, self-organising P2P video recorder to be developed by our research group. First, we pointed out how the "KISS" user interface design principle influences the requirements to the P2P protocols. After briefly describing two approaches for the scheduling component of the system, we have pointed out open issues. We believe that the practical experience with the system and its design will help to understand many aspects of peer-to-peer systems more thoroughly. It will probably also uncover some issues which have not been investigated by the research community yet.

References

[SSB04] Soursos, S., Stamoulis, G. D., and Bozios, T.: Distributed Scheduling of Recording Tasks with Interconnected Servers. In: *Proceedings of the Third International IFIP-TC6 Networking Conference*. p. 1483–1488. Athens, Greece. May 2004.

[WMM03] Walker, J., Morris, O. J., and Marusic, B.: Share it! A rights managed network of peer-to-peer set top boxes system architecture. In: *Proceedings of the International Broadcasting Convention (IBC) 2003*. Amsterdam. September 2003.

[YDIK98] Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*. 10(5):673–685. September 1998.